VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Absolvování individuální odborné praxe

Individual Professional Practice in the Company

2020 Dušan Hlaváč

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Dušan Hlaváč**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Absolvování individuální odborné praxe
Individual Professional Practice in the Company

Jazyk vypracování: čeština

Zásady pro vypracování:

1. Student vykoná individuální praxi ve firmě: CIT VŠB-TUO
2. Struktura závěrečné zprávy:
   a) Popis odborného zaměření firmy, u které student vykonal odbornou praxi a popis pracovního zařazení studenta.
   b) Seznam úkolů zadaných studentovi v průběhu odborné praxe s vyjádřením jejich časové náročnosti.
   c) Zvolený postup řešení zadaných úkolů.
   d) Teoretické a praktické znalosti a dovednosti získané v průběhu studia uplatněné studentem v průběhu odborné praxe.
   e) Znalosti či dovednosti scházející studentovi v průběhu odborné praxe.
   f) Dosažené výsledky v průběhu odborné praxe a její celkové zhodnocení.

Seznam doporučené odborné literatury:

Podle pokynů konzultanta, který vede odbornou praxi studenta.

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **doc. Mgr. Jiří Dvorský, Ph.D.**

Konzultant bakalářské práce: Ing. Pavel Rath

Datum zadání: 01.09.2019

Datum odevzdání: 30.04.2020

doc. Ing. Jan Platoš, Ph.D.
vedoucí katedry

prof. Ing. Pavel Brandštetter, CSc.
děkan fakulty

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 15. května 2020 Hlaváč .....

Souhlasím se zveřejněním této bakalářské práce dle požadavků čl. 26, odst. 9 Studijního a zkušebního řádu pro studium v bakalářských programech VŠB-TU Ostrava.

V Ostravě 15. května 2020

.....

Rád bych poděkoval všem svým kolegům z CIT VŠB-TUO za trpělivost, ochotu a příjemné pracovní prostředí. Zejména bych chtěl vyjádřit vděk Ing. Pavlu Rathovi za pomoc a vedení v průběhu celé praxe, Bc. Zdeňku Goldovi za mnoho užitečných rad a konzultací a v neposlední řadě také Ing. Martinu Lasoňovi za důvěru a vstřícnost.

Poděkování patří také vedoucímu práce doc. Mgr. Jiřímu Dvorskému, Ph.D., nejen za konstruktivní poznámky a pečlivou kontrolu textu práce, ale také za vytvoření LaTeX třídy Diploma, která mi ušetřila spoustu času.

Abstrakt

Tato bakalářská práce popisuje výkon mé individuální odborné praxe v oddělení informačních systémů Centra informačních technologií VŠB-TUO v průběhu posledního ročníku bakalářského studia na Fakultě elektrotechniky a informatiky. Postupně popíši úkoly, jejichž řešením jsem se zabýval, užité postupy, technologie a v neposlední řadě také problémy, které nevyhnutelně vznikly. V závěru se pokusím shrnout nabyté zkušenosti, znalosti, ať už technického či netechnického rázu, a výsledky své praxe.

Klíčová slova: Java EE, Spring Boot, Apache Maven, CAS, Alfresco, CIT VŠB-TUO, odborná praxe, webová aplikace

Abstract

This bachelor thesis describes my individual professional practice in the Department of Information Systems at Centre for Information Technology of VSB-TUO during the last year of my bachelor studies at the Faculty of Electrical Engineering and Computer Science. I will describe the tasks I dealt with, procedures used, technologies and also issues that inevitably arose. In the conclusion, I will attempt to summarise the gained experience, knowledge, be it technical or not, and results of my practice.

Keywords: Java EE, Spring Boot, Apache Maven, CAS, Alfresco, CIT VSB-TUO, professional practice, web application

Obsah

Seznam použitých zkratek a symbolů **9**

Seznam obrázků **10**

Seznam výpisů zdrojového kódu **11**

1 Úvod **12**
1.1 Motivace . . . 12
1.2 Volba firmy . . . 12
1.3 Očekávání . . . 12
1.4 Struktura bakalářské práce . . . 12

2 Profil firmy a popis pracovního zařazení **14**
2.1 Profil firmy . . . 14
2.2 Popis pracovního zařazení . . . 15

3 Použité technologie **16**
3.1 Java Enterprise Edition . . . 16
3.2 Apache Tomcat . . . 16
3.3 Apache Maven . . . 17
3.4 Git . . . 17
3.5 Apache Solr . . . 17
3.6 Central Authentication Service . . . 17
3.7 Alfresco . . . 18
3.8 Spring Framework . . . 18
3.9 Spring MVC . . . 18
3.10 Spring Security . . . 19
3.11 Spring Boot . . . 19

4 Infrastruktura **20**
4.1 Servery . . . 20
4.2 Aplikace . . . 20
4.3 Software pro vývoj . . . 20

5 Řešené úkoly a jejich časová náročnost **22**
5.1 Úkoly většího rozsahu . . . 22
5.2 Úkoly středního rozsahu . . . 23
5.3 Úkoly menšího rozsahu . . . 24

6 Řešení úkolů **25**
6.1 Přepsání komunikace s Apache Solr 25
6.2 Sjednocení metod a rozšíření centrální knihovny 26
6.3 Přesun knihoven z aplikací na server a jejich sjednocení 29
6.4 Rozdělení knihoven ze serveru do aplikací 30
6.5 Nasazení Apache Maven 33
6.6 Přepsání aplikace pro správu univerzitních dokumentů 35

7 Zhodnocení **39**
7.1 Uplatněné předchozí znalosti 39
7.2 Chybějící znalosti 39

8 Závěr **40**

Literatura **41**

Seznam použitých zkratek a symbolů

API	–	Application Programming Interface
CAS	–	Central Authentication Service
CIS	–	Centrum informačních služeb
CIT	–	Centrum informačních technologií
CN	–	Common Name
DI	–	Dependency Injection
DMS	–	Document Management System
DN	–	Distinguished Name
EJB	–	Enterprise JavaBeans
HTTP	–	Hypertext Transfer Protocol
HTTPS	–	Hypertext Transfer Protocol Secure
IDE	–	Integrated Development Environment
IoC	–	Inversion of Control
JAR	–	Java Archive
Java EE	–	Java Enterprise Edition
JDBC	–	Java Database Connectivity
JDK	–	Java Development Toolkit
JPA	–	Java Persistence API
JSON	–	JavaScript Object Notation
JVM	–	Java Virtual Machine
LDAP	–	Lightweight Directory Access Protocol
MVC	–	Model-View-Controller
OU	–	Organisational Unit
POM	–	Project Object Model
RDBMS	–	Relational Database Management System
RDN	–	Relative Distinguished Name
SLF4J	–	Simple Logging Facade for Java
SSO	–	Single Sign-on
TLD	–	Tag Library Descriptor
URI	–	Uniform Resource Identifier
URL	–	Uniform Resource Locator
VCS	–	Version Control System
WAR	–	Web Application Archive
XML	–	Extensible Markup Language

Seznam obrázků

1 Organizace serverů 20
2 Nové administrační rozhraní sklízeče dat pro fulltextové vyhledávání 26
3 Strom závislostí knihovny Spring ORM 30
4 Struktura cest pro logování událostí 32
5 Současný vzhled aplikace pro správu univerzitních dokumentů 35

Seznam výpisů zdrojového kódu

1 Přetěžování metod v centrální knihovně 28

1 Úvod

1.1 Motivace

Fakulta elektrotechniky a informatiky nabízí svým studentům posledního ročníku prezenčního bakalářského studia absolvování individuální odborné praxe jako alternativu k vypracování bakalářské práce. Využití této možnosti jsem zvažoval dlouhodobě již od počátku druhého ročníku, neboť praktické zkušenosti představují nespornou výhodu nejen na trhu práce, ale také při vypracovávání rozličných projektů a úloh na cvičeních i mimo ně. Zejména u netriviálních úkolů se ukázala být praxe v oboru neocenitelným pomocníkem.

1.2 Volba firmy

V průběhu hledání vhodného zaměstnavatele pro individuální odbornou praxi jsem absolvoval ještě před začátkem posledního ročníku několik pohovorů ve snaze najít si pracovní pozici přiměřené odbornosti, ideálně v oblasti back-end vývoje v jazyce Java s rozšířením Enterprise Edition. Existenci Centra informačních technologií Vysoké školy báňské - Technické univerzity Ostrava mi připomněl jeden z displejů panelového informačního obrazového systému v budově auly, na kterém spuštěná prezentace slibovala možnost spolupráce na bakalářské praxi. Toto se mi velmi zamlouvalo, protože jsem již při nástupu do prvního ročníku věděl, že si po dokončení studia budu hledat uplatnění primárně ve veřejném sektoru. Po úspěšném pohovoru jsem této nabídky využil a krátce na to nastoupil jako stážista - programátor webových aplikací se zařazením do týmu webových portálů v oddělení informačních systémů.

1.3 Očekávání

Od praxe jsem očekával především nabytí či prohloubení znalostí a zkušeností v oblasti vývoje webových aplikací, především pak v rámci konkrétních implementací v Java Enterprise Edition a Spring Framework. Vezmeme-li v úvahu také neméně důležité schopnosti netechnického rázu, chtěl jsem si zlepšit především schopnost týmové práce, komunikace, koordinační dovednosti a osvojit si zvyklosti specifické pro danou organizaci či kolektiv. Vzhledem k tomu, že se zároveň jednalo o mé první zaměstnání s vyžadovanou fyzickou přítomností na pracovišti, nemohu pominout také zisk jisté formy pracovních a mezilidských návyků s tím spojených a jejich zasazení do běžného dne v kanceláři.

1.4 Struktura bakalářské práce

V této bakalářské práci bych nejprve rád blíže popsal Centrum informačních technologií Vysoké školy báňské - Technické univerzity Ostrava a své pracovní zařazení v něm. Ve zbytku práce se budu věnovat použitým technologiím (přímo či nepřímo přispívajícím k vývoji webových aplikací), postupům odstranění některých vyvstalých problémů a úkolům, na jejichž řešení jsem se podílel.

V závěru stručně shrnu nabyté znalosti a schopnosti v oblasti informatiky, uvedu alespoň pár příkladů zlepšení kvality software pro koncové uživatele a pokusím se také zhodnotit, zdali se naplnila má očekávání či nikoliv. Příležitostně popíši krom zmiňovaných technických aspektů i jejich přesah do roviny tzv. měkkých dovedností (anglicky „soft skills“).

Veškeré informace ve formě textu doplním v případě potřeby taktéž ilustrací či jinou grafickou pomůckou, zejména části týkající se webových aplikací a práce s knihovnami, kde je vizuální reprezentace více než žádoucí.

2 Profil firmy a popis pracovního zařazení

2.1 Profil firmy

Centrum informačních technologií Vysoké školy báňské - Technické univerzity Ostrava (CIT VŠB-TUO) je celoškolským pracovištěm spadajícím pod Centrum informačních služeb (CIS) v úseku rektora. Pod CIS patří také Ústřední knihovna, Ediční středisko a Archiv. Všechny podútvary se společně zabývají tvorbou koncepcí, rozvojem a koordinací nasazení všech typů informačních technologií v rámci VŠB-TUO pro podporu úkolů spojených s předáváním informací svým „zákazníkům“ – studentům, akademickým pracovníkům, zaměstnancům VŠB-TUO, včetně široké veřejnosti [1].

Činnost CIT se soustředí primárně na tvorbu koncepce, implementaci a provoz informačních systémů školy spolu s návrhem a správou počítačové sítě. Mimo tyto úkoly poskytuje také poradenství pro fakulty a jiné útvary, poskytuje technickou podporu uživatelům systému a zajišťuje software nutný k jejich práci. Jedním z pověření je také provoz veškerého potřebného hardware pro výkon všech těchto povinností.
Centrum informačních technologií se dělí na tři oddělení:

- Oddělení informačních systémů (9871)
- Oddělení infrastruktury IT (9872)
- Audiovizuální služby (9873)

Oddělení informačních systémů čítá 26 zaměstnanců a zajišťuje provoz víc než dvou desítek systémů nezbytných pro chod univerzity, výuku, výzkum a vývoj. Patří mezi ně mimo jiné:

- EDISON – informační systém pro evidenci studia a výuky
- SAP – ekonomický systém
- InNET – portál pro zaměstnance a studenty
- Webový portál univerzity
- LMS – elektronický výukový systém
- Sylabus+ – systém pro tvorbu rozvrhů
- SafeQ – tiskový systém
- OBD – evidence publikací
- Kredit – stravovací systém
- Identity management pro správu uživatelů

Z dalších lze pak třeba jmenovat systém pro projekty IRP, elektronický platební systém, evidenci projektů (GAP), elektronický systém spisové služby a několik menších informačních systémů.

2.2 Popis pracovního zařazení

Do oddělení informačních systémů jsem nastoupil jako Java programátor se zařazením do týmu zabývajícího se webovými prezentacemi a portály. Po přijetí mi vedoucí přidělil pracovní místo a kolegové ukázali to nejnutnější do začátku. Rozšíření Enterprise Edition jazyka Java bylo pro mě novinkou, stejně tak systém pro správu úkolů Atlassian JIRA. S ostatními technologiemi jsem se naštěstí alespoň povrchně setkal již dříve, jejich užití tedy nepředstavovalo žádnou větší překážku. Krátce poté mi spolupracovníci vybrali první úkol, čímž zahájili mé působení v týmu.

3 Použité technologie

V této kapitole popíši technologie, se kterými jsem v rámci praxe pracoval nebo je využil k řešení zadaných úkolů.

3.1 Java Enterprise Edition

Java je čistě objektově orientovaný vysokoúrovňový programovací jazyk vyvinutý roku 1995 společností Sun Microsystems, který se těší popularitě zejména při práci na středních až velkých projektech. Nabízí jednoduchou syntaxi podobnou jazyku C a C++, automatickou správu paměti, rozsáhlý ekosystém knihoven a frameworků, open-source referenční implementaci a možnost spuštění na všech běžných platformách bez rekompilace. Multiplatformnosti dosahuje za pomocí JVM, což je virtuální stroj, na který může kompilátor cílit při převodu zdrojového kódu do Java bytecode. Odpadá tak nutnost přímého překladu do strojového kódu specifického pro danou architekturu, ale zároveň se zachovává většina výhod, jimiž disponují klasické kompilované jazyky v porovnání s interpretovanými. V současné době je Java nejoblíbenějším programovacím jazykem na světě [2].

Rozšíření Java Enterprise Edition (od konce roku 2019 Jakarta EE) je tvořeno sadou specifikací zaměřených primárně na webové aplikace a služby, především pak na jejich užití při implementaci podnikových informačních systémů. Mezi nejznámější součásti patří například Servlet, JavaServer Pages, Java Persistence API (JPA) nebo Enterprise JavaBeans (EJB).

3.2 Apache Tomcat

Apache Tomcat je open source software implementující technologie Java Servlet, JavaServer Pages, Java Expression Language a Java WebSocket [3]. Na rozdíl od plnohodnotných aplikačních serverů podporujících kompletní specifikaci Java Enterprise Edition slouží pouze jako Servlet kontejner, přestože umožňuje své rozšíření také o zbylé technologie. Obvyklá instalace se skládá z několika základních komponent:

- Coyote – konektorová komponenta starající se o podporu protokolu HTTP, umožňuje přijmout požadavek, předat jej ke zpracování a následně vrátit odpověď,
- Catalina – komponenta Servlet kontejneru, implementuje specifikace Java Servlet a JavaServer Pages,
- Jasper – komponenta JSP engine, spouští kompilaci nových JSP souborů a zajišťuje jejich rekompilaci při provedení změn.

Od verze Tomcat 7 jsou dále obsaženy také dodatečné komponenty pro klastrování, vysokou dostupnost a webové aplikace.

3.3 Apache Maven

Apache Maven je nástroj jehož nejznámějšími funkcemi jsou automatizace build procesu, správa závislostí, standardizace struktury projektu a poskytování srozumitelných informací o těchto činnostech [4]. Těchto cílů dosahuje, oproti svému předchůdci Apache Ant, pouze s minimem konfigurace na základě principu „convention over configuration“ v kombinaci s Project Object Model (POM). Jednu z hlavních výhod představuje centrální repozitář s již téměř 5 miliony artefakty, ke kterým Maven umožňuje snadný přístup, včetně automatického vyřešení konfliktů u tranzitivních závislostí a sestavení jejich stromu v případě potřeby manuálních úprav. Obdobně mocným nástrojem jsou také pluginy přinášející nesčetné možnosti přizpůsobení build procesu od jednoduchých až po velmi komplexní změny. Pro dodatečné zkrácení konfiguračního XML souboru mohou být projekty uspořádány do stromové struktury s dědičností jednotlivých nastavení.

3.4 Git

Git je nástroj pro distribuovanou správu verzí a v současnosti nejpopulárnější Version Control System (VCS) na světě [5]. Celý systém, výrazně se odlišující od svých předchůdců, byl vyvinut roku 2005 Linusem Torvaldsem pro usnadnění práce na jádře operačního systému Linux. Zaměřuje se především na rychlost, jednoduchost užití a podporu paralelizace vývoje ve velkém rozsahu. Z těchto důvodů probíhají téměř všechny operace lokálně, navíc s kontrolou integrity v každém kroku, a až po finálním potvrzení se synchronizují na vzdálený server, ke kterému více či méně pravidelně přistupují všichni uživatelé. Díky způsobu práce se soubory v Git je navíc možné vracet skoro všechny změny.

3.5 Apache Solr

Projekt Apache Solr je vyhledávací platforma nabízející možnost fulltextového vyhledávání, zvýrazňování hledané fráze nebo její části ve vrácených výsledcích, filtrování, indexování v reálném čase, dynamické klastrování, integraci databází a zpracování dokumentů v jiných formátech než prostý text (např. PDF, DOCX, ODT) [6]. Solr server je distribuován ve formě samostatné aplikace, která poskytuje API, dostupné přes HTTP protokol, podporující formáty XML a JSON. Místo přímé komunikace se obvykle nasazují klientské knihovny dostupné pro celou řadu populárních jazyků. Na pozadí Solr využívá knihovnu Apache Lucene, již sdílí i jiné projekty podobného zaměření (např. Elasticsearch).

3.6 Central Authentication Service

Central Authentication Service (CAS) je Single Sign-on (SSO) protokolem pro web. Umožňuje uživatelům přistupovat k několika aplikacím nebo službám pouze s jedinými jednotnými přihlašovacími údaji. Z pohledu aplikace je pak možné autentizovat uživatele bez potřeby znalosti jeho přístupovým údajů. První verze protokolu vznikla na přelomu tisíciletí a poměrně rychle se

rozšířila na stovky univerzit po celém světě (dnes již zřejmě tisíce). Procesu autentizace se obvykle účastní CAS server, CAS klient a aplikace požadující ověření. Krom prosté informace o úspěchu či neúspěchu přihlášení může server v odpovědi poskytnout i větší množství dodatečných atributů (např. role, osobní údaje) [7]. V případě naší univerzity se tyto informace získávají z LDAP. Typický proces úspěšné autentizace probíhá následovně (zjednodušeno):

1. Klient se pokusí o vstup do zabezpečené aplikace.
2. Vyvolá se přesměrování na přihlašovací stránku CAS.
3. CAS server ověří klienta, obvykle podle uživatelského jména a hesla.
4. CAS server přesměruje klienta zpět do aplikace a zároveň vrátí service ticket.
5. Klient ověří ticket kontaktováním CAS serveru přes zabezpečené spojení a poskytne vlastní ticket společně s identifikátorem služby.
6. CAS server vrátí klientovi informace o úspěšném přihlášení uživatele a případně také dodatečné atributy.

3.7 Alfresco

Alfresco je kolekcí software pro management firemního obsahu, business procesů a správu řízených dokumentů. V rámci praxe jsem se setkal pouze s částí DMS (Document Management System). Celá sada umožňuje mimo jiné také fulltextové vyhledávání, ukládání obrázků, automatizaci workflow, přidělování přístupových práv, zpracování a třízení vložených objektů a samozřejmě také verzování a sledování historie [8].

3.8 Spring Framework

Spring Framework slouží jako doplnění či kompletní náhrada některých částí Java EE specifikace, silně konkuruje především Enterprise JavaBeans. Jádro staví na principech Inversion of Control (IoC) a Dependency Injection (DI) [9]. Nejběžnějším případem užití je webová aplikace běžící v Servlet kontejneru nebo plnohodnotném aplikačním serveru. Nově je také možné framework využít pro reaktivní programování s pomocí Spring WebFlux a non-Servlet kontejneru (např. Netty nebo Undertow). Filosofií všech modulů Spring je snadná testovatelnost a minimalizace zbytečného kódu či nadměrného množství konfigurace, zvláště pak v XML, protože těmito problémy trpěla specifikace Java EE v době vzniku frameworku.

3.9 Spring MVC

Spring MVC (plným názvem Spring Web MVC) je jednou z původních součástí frameworku, která si klade za cíl usnadnit vývoj aplikací využívajících návrhový vzor Model-View-Controller s pomocí standardního Servlet API a následným nasazením do Servlet kontejneru. Podobně jako

spousta jiných frameworků, i Spring MVC implementuje běžný Front Controller Pattern [10]. Existuje tak jeden centrální vstupní bod se sdílenou logikou, který následně na základě jednoho či více faktorů (URI, HTTP metoda apod.) deleguje zpracování konkrétního požadavku dále na konkrétní komponenty. Díky této architektuře můžeme pokrýt s minimálními změnami širokou škálu případů užití.

3.10 Spring Security

Spring Security je framework usnadňující autentizaci a autorizaci uživatelů. V rámci praxe jsem jej použil v kombinaci s univerzitním CAS serverem poskutujícím SSO pro napojené aplikace. Krom standardních funkcí, jako například integrace s LDAP, CAS, podpora HTTP cookies a přidělování rolí či autorit, nabízí také operování přímo na úrovni webového serveru [11]. Tyto možnosti jsem se na návrh konzultanta pokusil využít jako náhradu CAS filtrů používaných v ostatních aplikacích, neboť ty neposkytovaly příliš elegantní řešení problému autentizace a autorizace uživatele. Bohužel však Spring Security zatím nepodporuje všechny součásti technologií, nad kterými tvoří vrstvu abstrakce. Jmenovitě jde například o Gateway funkcionalitu autentizačních CAS filtrů.

3.11 Spring Boot

Spring Boot je autokonfigurační nádstavba Spring Framework umožňující tvorbu aplikací a mikroslužeb kompletně bez XML konfiguračních souborů a deskriptorů nasazení. Preferovaným způsobem nasazení výsledného produktu je samostatný JAR soubor s přibalenou instancí Tomcat, Jetty nebo Undertow, ale podpora existuje i pro klasický WAR archiv v kombinaci s odděleným Servlet kontejnerem [12].

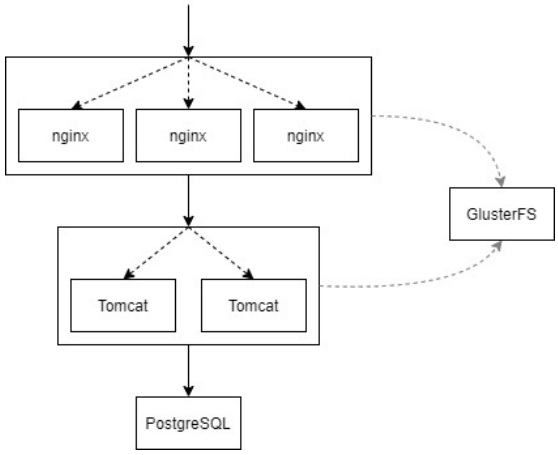
Filosofie celé nádstavby odpovídá populárnímu přístupu „convention over configuration“, který se uplatňuje například také u Apache Maven. Usnadnění a urychlení práce se týká i správy závislostí projektu, protože existují oficiální „startery“ nabízející předkonfigurované balíčky s kompatibilními verzemi knihoven pro určitou činnost (např. bezpečnost, logování, tvorba šablon pro uživatelské rozhraní). Pro produkční nasazení aplikace poskytuje framework metriky a kontroly zdraví bez nutnosti nastavování základních funkcí.

4 Infrastruktura

4.1 Servery

Infrastruktura obstarávající provoz webových prezentací a navázaných aplikací se skládá z většího množství vzájemně propojených částí (obrázek 1) s pevně definovanými rolemi. Požadavky se nejprve cyklicky rozdělují mezi tři nginx proxy servery, aby byla zátěž rovnoměrná. Ty pak směrují tok dat do klastru se dvěma aplikačními servery s instancemi Apache Tomcat napojenými na RDBMS PostgreSQL. Centralizované vysoce výkonné síťové úložiště poskytuje software GlusterFS.

K testování slouží dva nezávislé servery přístupné pouze z vnitřní sítě a oddělené databáze, jejichž obsah je pravidelně aktualizován na základě produkčního prostředí.



Obrázek 1: Organizace serverů

4.2 Aplikace

Pro provoz a editaci většiny prezentačních webových stránek a InNET se dlouhodobě využívá systém pro správu obsahu OpenCms s několika proprietárními doplňky. Tento software je nasazen na serverech s instancí Servlet kontejneru Apache Tomcat. Ostatní aplikace, v podobě WAR archivů, disponují vlastní nezávislou funkcionalitou a často také poskytují data formou exportů. Ke konci praxe proběhl upgrade z Java 7 na verzi 8, čímž se vyřešily potíže s kompatibilitou novějších knihoven.

4.3 Software pro vývoj

Existují celkem tři Git repozitáře, se kterými se při vývoji obvykle pracuje:

- opencms – OpenCms moduly a dodatečné JSP soubory,
- unisweb – hlavní repozitář obsahující aplikace a konfiguraci serveru,

- waal – autentizační aplikace WAAL.

Poslední dva jmenované jsou zcela nezávislé, ale OpenCms moduly vyžadují některé soubory z hlavního repozitáře *unisweb*. Mimo jiné také z tohoto důvodu u něj, na rozdíl od zbytku projektů, nelze využít Apache Maven.

5 Řešené úkoly a jejich časová náročnost

V průběhu své individuální odborné praxe, jsem byl řešitelem 23 úkolů, které svým rozsahem pokrývají spektrum od drobných až po relativně rozsáhlé. Zhruba polovinu až dvě třetiny času jsem se věnoval práci na zadaných úkolech, zbylý čas pak řešení těch vytvořených z vlastní iniciativy. Ve výsledku se provedené změny ve větší či menší míře dotkly všech užívaných aplikací.

Níže uvedené časové údaje jsou skutečně jen velmi hrubým odhadem založeným na přibližných údajích v JIRA, protože přesnější výpočet nebyl z různých důvodů možný. Pojmem „den“ se tedy myslí den kalendářní, nikoli ten pracovní, jehož délka v průběhu praxe fluktuovala. Kvůli práci na více úkolech současně pak také nemusí odpovídat jejich celkový součet.

5.1 Úkoly většího rozsahu

- Přepsání komunikace s Apache Solr (20 dnů)
  - Analýza původní verze: 3 dny
  - Hledání a analýza vhodného řešení: 2 dny
  - Sestavení minimální funkční testovací verze: 2 dny
  - Přepsání vyhledávací funkcionality: 3 dny
  - Přepsání sklízecí funkcionality: 3 dny
  - Úprava administračního rozhraní: 1 den
  - Standardizace komunikace s plánovačem úloh Quartz: 2 dny
  - Oprava nalezených chyb: 2 dny
  - Oddělení sklízení do samostatného vlákna: 1 den
  - Refactoring, dokumentace a drobné úpravy: 1 den
- Sjednocení metod a rozšíření centrální knihovny (14 dnů)
  - Analýza obecných a duplicitních metod: 2 dny
  - Sepsání seznamu metod k nahrazení a míst jejich užití: 2 dny
  - Vytvoření nových metod: 3 dny
  - Nahrazení staré implementace novou: 3 dny
  - Příprava testovacího prostředí: 2 dny
  - Testování a odstranění nalezených chyb: 2 dny
- Přesun knihoven z aplikací na server a jejich sjednocení (6 dnů)
  - Analýza všech knihoven: 2 dny
  - Nalezení duplicitních knihoven: 2 dny

  - Sestavení seznamu pro sjednocení: 1 den
  - Přesun části knihoven: 1 den

- Rozdělení knihoven ze serveru do aplikací (16 dnů)
  - Analýza závislostí sdílených knihoven: 2 dny
  - Zpracování návrhů řešení úkolu: 1 den
  - Rozdělení knihoven do aplikací: 4 dny
  - Testování: 3 dny
  - Odstraňování nalezených problémů: 2 dny
  - Sjednocení XML konfiguračních souborů: 1 den
  - Přepsání administračních rozhraní do nového vzhledu: 1 den
  - Refactoring a dokončovací práce: 2 dny

- Nasazení Apache Maven (11 dnů)
  - Analýza možných řešení: 2 dny
  - Návrh finálního řešení: 1 den
  - Příprava konfigurací: 3 dny
  - Převod aplikací na Maven: 1 den
  - Testování: 2 dny
  - Sepsání dokumentace: 1 den
  - Refactoring a dokončovací práce: 1 den

- Přepsání aplikace pro správu univerzitních dokumentů (32 dnů)
  - Analýza dostupných technologií: 1 den
  - Vytvoření šablony Spring Boot aplikace: 3 dny
  - Analýza struktury původní aplikace: 1 den
  - Implementace a konfigurace zabezpečení: 4 dny
  - Vytvoření konvertorů tříd modelů: 2 dny
  - Přepsání vrstvy služeb: 21 dnů

5.2 Úkoly středního rozsahu

- Nahrazení „boostů“ při indexování do Apache Solr (4 dny)
  - Analýza dostupných řešení: 1 den

  - Přepsání indexování a vyhledávání: 1 den
  - Přepsání administračního rozhraní do nového vzhledu: 1 den
  - Refactoring a testování: 1 den
- Sjednocení formátování kódu (4 dny)
  - Analýza výchozích nastavení IDE: 1 den
  - Vytvoření sdílené konfigurace: 1 den
  - Revize zdrojových kódů a sjednocení formátování: 1 den
  - Testování: 1 den

5.3 Úkoly menšího rozsahu

Celkový čas zpracování jeden den (dohromady 12 dnů):

- Automatický monitoring dostupnosti vybraných dokumentů
- Přidání možnosti odstranění dodatečných jazykových mutací dokumentu
- Mazání neověřených absolventských účtů v aplikaci Alumni
- Úprava procesu mazání přílohy dokumentu
- Úprava vzhledu exportu dokumentů na webu
- Ošetření výjimek v aplikaci pro správu univerzitních dokumentů
- Úprava zástupného textu při vyhledávání v telefonním seznamu
- Ošetření problémů s databázovým spojením v aplikaci WAAL
- Odstranění závislosti aplikace WAAL na obsahu jiného repozitáře
- Oprava chybové stránky aplikace pro správu univerzitních dokumentů
- Převedení konektoru pro Alfresco DMS na Maven
- Zprovoznit Maven projekt Alfresco konektoru pro SAP

Celkový čas zpracování dva dny (dohromady 6 dnů):

- Ukládání vyplněných textů závěrečných zpráv z mobilit
- Zobrazení popisu a velikosti souboru u příloh dokumentů na webu
- Odstranění prázdných řádků ve výstupu z JSP souborů

6 Řešení úkolů

V této kapitole se budu věnovat konkrétnímu popisu řešení úkolů, vzniklých problémů a postupů užitých k jejich odstranění.

6.1 Přepsání komunikace s Apache Solr

Cílem tohoto úkolu bylo standardizovat komunikaci s instancí Apache Solr v aplikaci zprostředkovávající fulltextové vyhledávání a sklízení pro webové portály školy.

V původní verzi aplikace pro vyhledávání a sklízení probíhala veškerá interakce ve formátu XML, jenž svým obsahem redundantních informací zbytečně zvyšuje vytížení sítě a zpomaluje zpracování dat. Bez využití knihovny SolrJ se navíc stalo programové generování XML dokumentů nevyhnutelným, což značně přispělo ke vzniku velkých, místy nepřehledných, kusů kódu.

Upravená aplikace vzniklá dokončením úkolu již využívá výhod oficiální knihovny SolrJ. Veškerá komunikace probíhá v binárním formátu javabin, navíc s pomocí automatického mapování vrácených výsledků na Java objekty.

6.1.1 Analýza

Nejprve bylo nutné seznámit se s aplikací a tokem dat oběma směry v ní. Tato analýza mě bohužel poněkud zdržela, neboť zdrojový kód obsahuje několik vrstev abstrakce a řadu rozhraní zprostředkovávajících vyhledávání pro externí služby. U některých částí aplikace se mi dokonce ani nepodařilo dohledat k čemu slouží nebo zdali se ještě vůbec využívají.

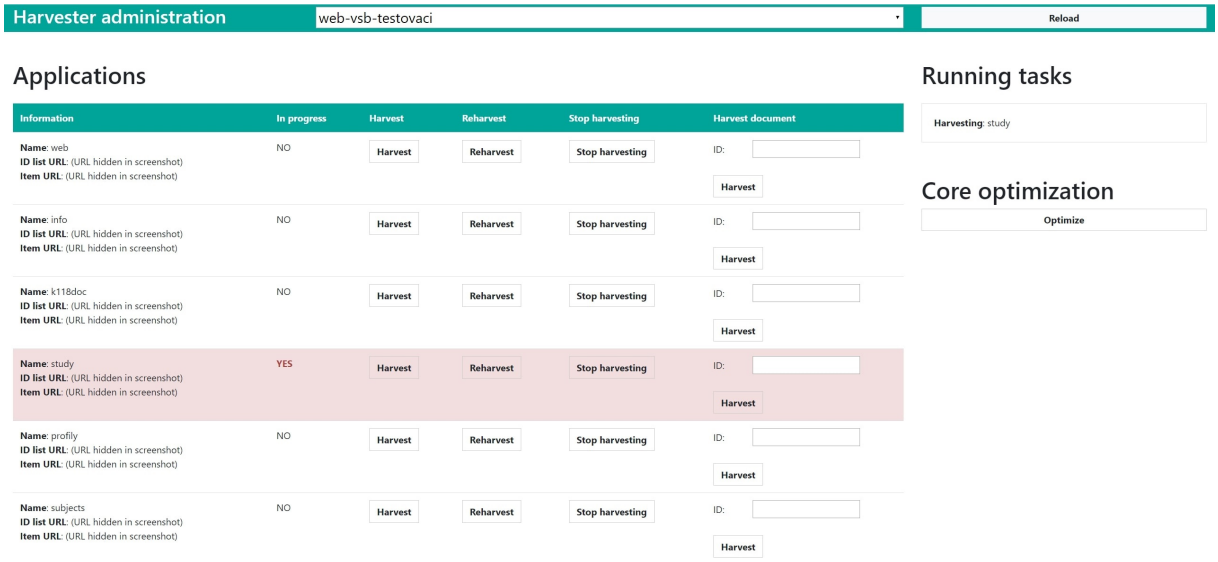
Jako součást zadání mi také konzultant doporučil ověřit si funkčnost zabezpečené komunikace po vnitřní síti s instancí Apache Solr přes protokol HTTPS. Při plnění tohoto podúkolu jsem narazil na menší překážku v podobě požadavku minimálně Java 8 u aktuální knihovny SolrJ. Naštěstí jsou i starší verze SolrJ dopředně kompatibilní, což mi umožnilo využít jednu z těch podporujících Java 7, ověřit si funkčnost komunikace a pokračovat v řešení úkolu.

6.1.2 Přepsání sklízecí a vyhledávací funkcionality

Z knihovny SolrJ jsem pro výrazné zkrácení a zobecnění kódu využil schopnost mapovat vrácené výsledky na Java objekty pomocí anotací. Z důvodu zachování současné funkcionality bylo i přesto nutné umožnit serializaci objektu do XML, nicméně tohoto se mi povedlo dosáhnout bez použití externích knihoven, které by se při budoucí aktualizaci aplikace musely odstraňovat.

Dalším logickým krokem bylo přepnutí všech operací tak, aby využívaly efektivnější komunikaci přes knihovnu SolrJ. Tato část úkolu již probíhala poměrně hladce. Kombinací úspornějšího binárního formátu, optimalizace kódu a využití vhodné knihovny se doba pravidelného sklízení zkrátila odhadem o více než 40 %, což v reálném nasazení představuje úsporu několika minut.

Konzultant mě upozornil, že administrační rozhraní v rámci svého provozu vždy čekalo na dokončení operace. Toto chování nicméně způsobovalo téměř jisté vypršení spojení, protože akce



Obrázek 2: Nové administrační rozhraní sklízeče dat pro fulltextové vyhledávání

v tomto případě na serveru běží standardně i několik minut. Pro odstranění těchto potíží jsem zvolil přímočarou cestu spouštění běžící úlohy v samostatném vlákně. Při příležitosti úprav v aplikaci mi bylo dovoleno si osvěžit znalosti návrhu responzivní webové stránky a upravit administrační rozhraní do moderního vzhledu inspirovaného novým vizuálním stylem univerzity (obrázek 2).

6.1.3 Standardizace komunikace s Quartz

Konzultant mě dodatečně požádal, abych upravil komunikaci s Quartz do standardní podoby spočívající v přístupovém bodu s autentizací, který po přijmutí požadavku spustí požadovanou úlohu na specifikovaném uzlu v klastru. Díky vyřešení tohoto podúkolu bylo možné odstranit z aplikace knihovnu pro Quartz, jež původně spouštěla operace nezávisle na centrální konfiguraci.

6.2 Sjednocení metod a rozšíření centrální knihovny

Tento úkol si kladl za cíl minimalizaci duplicit v kódu v podobě metod, které se nacházely v několika aplikacích zároveň, přestože sloužily identickému či velmi podobnému účelu.

Původní zadání vytvořil jeden z kolegů, který si tohoto problému povšiml již dříve. Řešení se samozřejmě dotýkalo značné části zdrojových kódů napříč dvěma repozitáři. Tento důvod si, společně s řadou dalších, vyžádal rozsáhlou analýzu, již z nemalé části nešlo automatizovat. Po dokončení se do testování zapojil prakticky celý tým, abychom včas zachytili případné nedostatky a opravili je ještě před nasazením do ostrého provozu. Do zkoušky některých aplikací jsem se nemohl zapojit, neboť pro přístup vyžadovaly osobu s vyšším oprávněním či specifickou rolí.

Plný rozsah výhod rozšířené centrální knihovny se pochopitelně projeví především v budoucnu, ať už při údržbě a rozšiřování současných aplikací nebo tvorbě zcela nových.

6.2.1 Analýza

Nejprve jsem sestavil seznam duplicitních metod a zakomponoval do něj i několik existujících položek nalezených kolegou. Tuto část bohužel nešlo nijak automatizovat, protože existující metody měly nejednotné nejen názvy, ale také typ a pořadí přijímaných parametrů, navíc se také v některých případech částečně lišila jejich implementace.

Za druhé jsem se rozhodl sestavit seznam metod, které díky své obecné povaze mohou být využity na mnoha místech, avšak doposud se nacházely pouze přímo v jedné z aplikací. Nezbytné úkony rovněž nebylo možné automatizovat.

S konzultantem jsme se oba shodli na rozdělení nových metod do jednotlivých tříd dle účelu, přičemž při vytváření struktury se inspirujeme velmi známou a populární knihovnou Apache Commons Lang 3. Výsledkem bylo zaobalení statických „utility“ metod do devíti tříd:

- AuthenticationUtils – autentizace a úkony s ní přímo související
- CollectionUtils – kolekce a práce s nimi
- DateUtils – datum a čas
- FileUtils – manipulace se soubory
- MultipartUtils – zpracovávání HTTP požadavků skládajících se z více částí
- ServletRequestUtils – extrahování informací ze Servlet požadavku
- StringUtils – práce s textovými řetězci
- ValidationUtils – validace
- WebUtils – obecné metody pro práci s webem

Výsledný seznam duplicit čítal celkem 19 základních typů metod s větší či menší shodou v implementaci v podobě 56 konkrétních metod ve 14 aplikacích, přičemž všechny měly povahu utilit. Přesunul jsem také z 11 modulů dalších 21 metod, které sice nebyly zdvojené, nicméně se díky své obecnosti jevily být dobrými kandidáty na širší využití v budoucnu. V ojedinělých případech si změny vyžádaly také přidání či odebrání knihovny.

6.2.2 Rozšiřování centrální knihovny

Za využití seznamů vytvořených v průběhu analýzy jsem postupně přidával metody do nově vytvořeného balíčku v centrální knihovně. V nezanedbatelném množství případů se však ukázalo,

že zdánlivě identické metody se liší v typu a pořadí přijímaných parametrů či dokonce v implementaci. Z tohoto důvodu bylo nutné ke každé z nich přistupovat individuálně a pokusit se navrhnout nové „utility“ tak, aby se nezměnilo jejich chování, ale přesto mohly být přesunuty a sjednoceny.

Jazyk Java bohužel neumožňuje specifikovat výchozí hodnotu parametrů metody. Tento nedostatek se běžně obchází několikanásobnou definicí metody v rámci třídy, z nichž každá představuje jinou variantu z hlediska počtu parametrů či jejich datových typů (výpis 1). Při volání se následně automaticky zvolí vhodně přetížená verze metody. Tohoto postupu jsem při návrhu centrální knihovny hojně využíval pro zlepšení čitelnosti kódu a pokrytí maximálního možného počtu scénářů.

```
public static List<String> parseStringList(String input, String delimiter) {
   if (input != null) {
      if (input.length() > 0) {
         return Arrays.asList(input.split(delimiter));
      } else {
         return new ArrayList<>();
      }
   } else {
      return null;
   }
}

public static List<String> parseStringList(String input) {
   return parseStringList(input, ",");
}
```

Výpis 1: Přetěžování metod v centrální knihovně

6.2.3 Nahrazení stávajících metod novými

Po dokončení rozšiřování jsem nahradil stávající metody těmi nově vytvořenými. V rámci klasického Java kódu šlo vše bez problémů, nicméně v JSP souborech se mi nepovedlo zprovoznit v IDE automatickou validaci při kompilaci projektu. Z této nepříjemnosti při testování plynula nutnost manuálně otevřít každé JSP a počkat až několik sekund na dokončení validace. Tento postup jsem pak musel opakovat také před sloučením vedlejší Git větve pro tento úkol s tou hlavní, abychom se ujistili, že nahrazení proběhlo v pořádku.

6.3 Přesun knihoven z aplikací na server a jejich sjednocení

Tento úkol vznikl z mé vlastní iniciativy a zaměřil se na knihovny v aplikacích a jejich přesun do sdíleného umístění na server, čímž by zároveň doslo ke sjednocení verzí. Hlavní výhodu by představovalo zrychlení načítání po startu Apache Tomcat a zjednodušení správy závislostí. Vzhledem k nemalému rozsahu plánovaných změn jsem realizaci rozdělil do několika navazujících fází:

1. analýza rozložení knihoven,
2. sběr informací o verzích a závislostech,
3. přesun knihoven,
4. doplnění runtime závislostí.

6.3.1 Analýza, sběr a třídění informací

Za účelem usnadnění realizace první a druhé fáze jsem si vytvořil pomocnou SQLite databázi pro evidenci knihoven, jejich verzí a rozmístění v aplikacích. Vzniklé tabulky popisovaly celkem 542 knihoven a jejich závislostí, z toho 124 duplicit. Tyto informace se však ukázaly být nedostatečné, neboť z nich nešlo sestavit stromovou strukturu reprezentující vztahy mezi knihovnami. Pro doplnění jsem využil POM soubory z centrálního repozitáře Apache Maven procházené pomocí kombinace online nástrojů a IDE.

6.3.2 Přesun knihoven

Ve třetí fázi došlo k přesunu pouze několika málo knihoven. Krátce po započetí práce jsem totiž zjistil, že vyřešení celého úkolu je nejen nereálné, ale také zcela kontraproduktivní z hlediska budoucích plánů. Někteří autoři software navíc bohužel stále nedodržují pravidla sématického verzování, což celý proces jen dále ztížilo.

6.3.3 Příčiny předčasného ukončení řešení

Ukázalo se, že sjednotit některé knihovny není vůbec možné, protože by tím zároveň došlo k upgrade verze ve starších aplikacích, což by mělo za následek jejich nefunkčnost. Zejména u Spring Framework a Hibernate je pak naprosto nezbytné, aby mohlo vedle sebe běžet několik nezávislých verzí. V určitých případech navíc vznikl konflikt verzí, který za předpokladu společného umístění knihoven nešlo vůbec vyřešit. Pro plánované nasazení Apache Maven by dokončení tohoto úkolu navíc znamenalo značné navýšení obtížnosti, neboť závislosti aplikací je ve většině případů nutno spravovat separátně. Fakt, že se obecně doporučuje mít ve sdíleném umístění pouze minimum položek a nezbytnost restartu služby Apache Tomcat po každé jejich změně již jen potvrdil nesprávnost původní myšlenky.

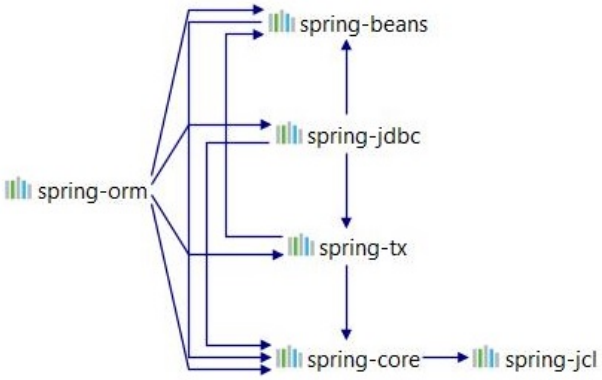
6.4 Rozdělení knihoven ze serveru do aplikací

Cílem úkolu bylo rozmístit knihovny do jednotlivých aplikací, čímž bychom minimalizovali využití sdíleného umístění a umožnili tak oddělenou správu závislostí. Dokončení těchto kroků následně otevřelo cestu ke sjednocení konfigurace projektu a odstranění prvků specifických pro Eclipse IDE, které ztěžovaly import a práci s moduly v jiných vývojových prostředích. Těchto výhod jsme v jednom z dalších úkolů dosáhli nasazením Apache Maven a kompletním vytěsněním IDE z procesu kompilace a sestavení výsledného archivu. Nyní se pro tyto úkony využívá konfigurace specifikovaná v POM souborech, jež pracuje se standardním Java kompilátorem z JDK.

6.4.1 Analýza

Před započetím práce jsem znovu sestavil aktuální seznam knihoven na serveru i v aplikacích a pokusil se také určit závislosti mezi nimi. U položek veřejně dostupných v Maven repozitářích nepředstavovalo získání takových informací větší problém, byť bylo, alespoň před optimalizací procesu, časově náročné. Knihovny z IBM WebSphere nezbytné pro funkčnost EJB z informačního systému EDISON však volně k dispozici nejsou a dohledat jejich závislosti si tak vyžádalo časově neefektivní metodu pokus – omyl s testováním po každé změně. Práci také dodatečně ztěžoval fakt, že absence některých knihoven se neprojevila okamžitě, nýbrž pouze při nasazení nebo dokonce až v průběhu testování aplikace.

K provedení analýzy knihoven v jednotlivých modulech projektu na základě jejich využití ve zdrojovém kódu jsem se rozhodl uplatnit analyzátor závislostí v IDE, který dokáže sestavit seznam včetně údaje o umístění nezbytného pro rozlišení mezi JAR souborem na serveru a v aplikaci. Výstup jsem v několika případech namátkově ověřil a odpovídal skutečnosti. Mimo jiné lze nástrojem také snadno vizualizovat strom artefaktů z Apache Maven (obrázek 3). Rozdíl mezi touto množinou a výsledkem analýzy v IDE pak představoval s největší pravděpodobností většinu tranzitivních závislostí knihoven.



Obrázek 3: Strom závislostí knihovny Spring ORM

Preventivně jsem také zjistil možnosti řešení konfliktů balíčků v novějších verzích jazyka Java pro případ problémů s tzv. „fat JAR“ (archiv obsahující kromě zkompilovaného zdrojového kódu

také nezbytné knihovny) a došel k závěru, že Project Jigsaw a jeho modulový systém přidaný ve verzi Java 9 má s největší pravděpodobností dostatečné prostředky k vyřešení překážek tohoto typu.

6.4.2 Přesun knihoven

Po jednom manuálním pokusu a diskuzi s konzultantem jsem se rozhodl pro částečně automatizované řešení s Apache Maven, jež ušetří práci nejen v rámci tohoto úkolu, ale také při plánované konverzi aplikací. Při rozdělování se pro každou aplikaci sestavila POM konfigurace se zásuvným modulem pro nakopírování všech závislostí do konkrétní složky. Po jejím spuštění pak stačilo porovnat výsledek s původním stavem a doplnit případné chybějící knihovny. V některých případech nastal problém se závislostmi, které neměly žádné explicitní využití v kódu a zároveň je repozitář označil za volitelné. U takových aplikací bylo nutné dodatečně otestovat také různé kombinace těchto knihoven a zjistit tak nezbytnost jejich přítomnosti.

Krom nutného minima byly na serveru z důvodu způsobu zavádění do paměti ponechány archivy obsahující JDBC implementace. U části knihoven jsem provedl upgrade na novější verze, protože ty původní nešlo dohledat v repozitářích Apache Maven či se jednalo o nestandardní distribuci. U velmi malé části závislostí došlo z důvodu nadbytečnosti k úplnému odstranění. Speciálním případem byla existující instalace OpenCms, která vyžadovala individuální přístup ke každé přidávané knihovně, aby nedošlo k nežádoucímu upgrade či downgrade verze nebo konfliktu balíčků, přičemž jakákoli změna vyžadovala restart Apache Tomcat.

Při řešení tohoto úkolu jsem narazil mimo jiné také na problém vzniklý kombinací specifického způsobu načítání knihoven v Apache Tomcat a identifikací tříd v jazyce Java. Běžně se tzv. classloadery větví do stromové hierarchie, což ovšem v případě webových aplikací není zcela žádoucí, protože závislosti nacházející se ve WAR souboru by měly mít přednost před těmi na serveru. Z tohoto důvodu Servlet kontejner invertuje pořadí posledních dvou typů classloaderů a načítá knihovny z aplikací před těmi společnými. Za shodné jsou pak na úrovni JDK považovány třídy, jež mají identický nejen název a balíček, ale také classloader, který je načetl. V případě jedné ze závilostí jsem tak narazil na to, že nelze v metodě vracet instanci třídy načtené ze společného umístění a následně ji zpracovat v jedné z aplikací, jež má ale přiloženou svou vlastní kopii knihovny.

Změny dosažené dokončením tohoto úkolu se dotkly všech 24 aplikací a více než 400 knihoven. Na testování se kromě mě podíleli také tři další kolegové. Několik málo změn muselo být při testování vráceno zpět, neboť se ukázalo, že neumožňují správnou funkci EJB pro integraci s IS EDISON a souvisejících IBM WebSphere knihoven.

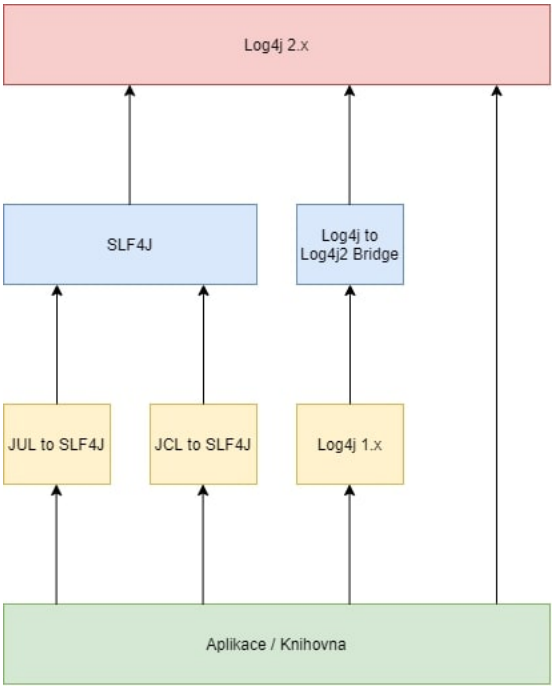
6.4.3 Ostatní podúlohy

Krom přesunu knihoven se do řešení vmísily také jiné drobnější opravy a úpravy. Přepsal jsem jedno z administračních rozhraní do moderního vzhledu inspirovaného novým vizuálním stylem

univerzity, opravil kosmetické nedostatky v těch již modernizovaných a sjednotil deskriptory nasazení všech aplikací tak, aby obsahovaly pouze validní XML a uvedené verze schémat odpovídaly aktuálnímu stavu Servlet kontejneru. Díky druhé zmiňované části se mimo jiné zpřístupnily některé novější prvky standardu JSP, čímž zmizela zbytečná varování a zlepšila se schopnost automatického doplňování v IDE.

Upravil jsem také několik definic filtrů či Servletů a realizoval drobnější opravy v TLD souborech, díky čemuž tyto části odpovídají specifikaci Java EE a doporučeným postupům. Přestože kolegové IntelliJ IDEA zatím nepoužívají, zdálo se mi vhodné zpracovat, nejen pro vlastní potřebu, dokumentaci pro práci s projekty, neboť se v současné době jedná o nejrozšířenější Java IDE [13] [14] a byla by škoda ověřené postupy nezaznamenat.

Z rozdělení knihoven také přirozeně vyplynula potřeba sjednotit závislosti pro logování událostí, protože jich existuje celá řada a součástí aplikace jich obvykle bývá několik, přestože se ve vlastním kódu využívá jen jedna. Vzhledem k plánovanému nasazení SLF4J navíc bylo vhodné směrovat do centrální Log4j 2 implementace veškeré události, nejen ty námi generované. Umístěním tzv. „bridge“ knihoven pro logování do úložiště načítaného sdíleným classloaderem jsem usměrnil veškeré zprávy do fasády, jejich následným odstraněním z aplikací se předešlo nekonečnému zacyklení při přesměrování (obrázek 4).



Obrázek 4: Struktura cest pro logování událostí

6.5 Nasazení Apache Maven

Tento úkol logicky vyplynul z rozdělení knihoven do aplikací jako další a finální krok směrem k automatizaci správy závilostí a eliminaci IDE z procesu sestavení. Převod konfigurace do podoby POM souboru a následné využití Apache Maven s sebou přineslo řadu výhod, zejména pak v oblasti usnadnění budoucích aktualizací a změn. Řešení úkolu si vyžádalo změny ve všech 23 existujících aplikacích a také nastavení několika stovek závislostí.

Původní přístup spoléhal na uložení projektových souborů z Eclipse přímo do Git repozitáře, přičemž navíc závisel na lokální instalaci Apache Tomcat, do které se nakopírovala sdílená část knihoven. Z toho pochopitelně plynula nejen nevýhoda podstatně složitějšího manuálního importu modulů v jiném IDE, ale také nutnost vytvořit zcela odlišnou konfiguraci v případě absence zmiňované lokální Tomcat instance.

V průběhu celé práce na úkolu jsem bohužel musel několikrát změnit přístup, neboť informace dostupné online jsou ze značné části zastaralé nebo nabádají k nesprávným či příliš komplikovaným řešením. Původně jsem taktéž vycházel z mylné domněnky, že nemohu nijak měnit kód aplikací, což při jejich rozložení do dvou, ne zcela nezávislých, Git repozitářů vedlo k velmi nešikovnému výsledku prvotní analýzy. Separátní projekt s moduly a JSP pro OpenCms jsme se již na začátku rozhodli zcela vynechat, protože neobsahuje žádnou ucelenou aplikaci.

6.5.1 Analýza

Bohužel jsem si neponechal jednotlivé verze POM souboru použitého při rozdělování knihoven do aplikací, bylo tedy nutné znovu zrevidovat kompletní seznam závislostí čítající přes 400 položek a začít v tomto ohledu od nuly. S výjimkou krátkého časového úseku hned z kraje analýzy se jevila varianta vícemodulového Maven projektu jako jediná rozumná.

Konzultant původně navrhoval hybridní řešení, kdy sestavení zůstává na starosti IDE a správu závislostí obstarává Apache Maven. V průběhu analýzy jsme ale tuto variantu brzy oba zavrhli a při zpětném pohledu si nejsem vůbec jistý, zda by ji šlo realizovat. Návrhů postupu následně vzniklo hned několik, přičemž výsledek připomíná ten původní jen minimálně.

Ve všech postupně navrhovaných variantách konfigurace představovaly primární překážku interní knihovny, které měly všechny uvedeny shodnou verzi nebo nebyly verzované vůbec. Podpora Apache Maven pro práci s JAR archivy fyzicky umístěnými přímo na souborovém systému vývojářova počítače obecně není příliš dobrá, protože takové postupy přímo odporují filosofii platformy. Kombinace těchto problémů si tak vyžádala specifické řešení šité na míru přímo našemu případu užití. Nutno podotknout, že ideálním řešením je v takové situaci instalace vlastního podnikového repozitáře (nejčastěji Nexus nebo Artifactory), nicméně z důvodu malého počtu uživatelů i interních knihoven by se nám tento postup nevyplatil.

6.5.2 Navrhované varianty rozložení projektu

Prvotní návrh počítal s konkrétním fyzickým rozmístěním naklonovaných Git repozitářů, aby šlo přistupovat k interním knihovnám pomocí relativní cesty. V návaznosti na tento krok jsem měl v úmyslu napsat skript, který by po spuštění nainstaloval předem dané závilosti do lokálního Maven uživatelského úložiště. Rodičovský projekt by se pak nacházel pouze v hlavním Git repozitáři, přičemž ten vedlejší bychom považovali pouze za modul. Toto řešení úkolu jsem pokládal za krajně nevhodné, nicméně jediné proveditelné beze změn v kódu. Po diskuzi s konzultantem se ale ukázalo, že takové omezení není na místě a tento přístup tak ztratil smysl.

Druhá varianta se již dosti podobala finální verzi, ale počítala s umístěním lokálního Maven repozitáře pro interní knihovny přímo ve složce projektu. Hlavní výhoda tohoto řešení spočívala v možnosti sdílení aktuálního stavu úložiště pomocí VCS, díky čemuž by stačilo, aby při aktualizaci závislostí provedl instalaci jen jeden ze členů týmu. Ukázalo se však, že i takový přístup trpí celou řadou neduhů, které v mnoha ohledech ztěžují či znemožňují jeho užití v praxi. Největší překážku představovalo odkazování se na JAR soubory interních knihoven pomocí relativních cest, jež se však nechovaly dle očekávání při přístupu v modulech.

Jediná změna v rozložení vedoucí k finálnímu návrhu spočívala v odstranění posledního nedostatku v podobě lokálního Maven repozitáře ve složce projektu. Odpadly tak problémy s cestou k archivům s knihovnami, zpřehlednila se konfigurace, nepatrně se zrychlily Git operace a také se zabránilo zbytečnému duplikování JAR souborů ve VCS. Veškeré interní závislosti se nyní instalují do standardního uživatelského Maven repozitáře v počítači vývojáře, přičemž zdrojem pro tuto akci jsou jejich aktuální verze ve zvláštní složce v projektu.

6.5.3 Práce s projektem

Postupem času samozřejmě vzniká potřeba některou z interních knihoven aktualizovat. Pro tyto případy jsem přidal do konfigurace dva zásuvné moduly a spojil je s fázemi životního cyklu procesu sestavení. První zajišťuje smazání současných verzí závislostí z lokálního uživatelského repozitáře v průběhu fáze „clean“. Ten druhý pak obstarává v rámci fáze „validate“ instalaci aktualizovaných JAR souborů z pevně daného umístění v projektu.

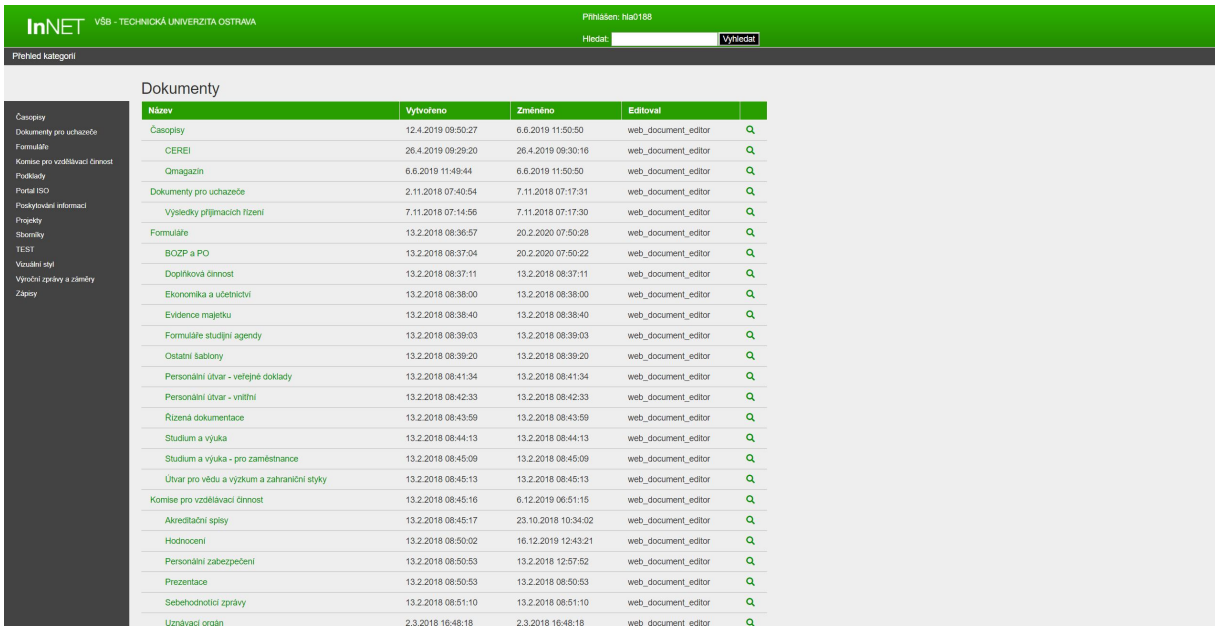
Rozšiřitelnost Apache Maven lze samozřejmě využít i při běžné práci. Pro snadnou identifikaci nasazené verze aplikace jsem přídal plugin pro zaznamenání aktuálního Git commitu při sestavení do souboru MANIFEST.MF, který je automaticky přikládán do složky META-INF každého WAR i JAR archivu. Nutnost vytvořit z centrální knihovny tzv. „fat JAR“ nás také přiměla využít zásuvný modul umožňující přiložení a rozbalení závislostí do výsledného artefaktu. Poslední přidaný plugin již jen do manifestu doplňuje autora sestavení a některé diagnostické informace (např. verzi JDK či operační systém). Bohužel ani při hledání těchto rozšíření jsem se nevyhnul zastaralým či nesprávným doporučením u jinak poměrně důvěryhodných online zdrojů, což mě místy dosti zdrželo.

6.6 Přepsání aplikace pro správu univerzitních dokumentů

Tento úkol si kladl za cíl kompletně přepsat aplikaci pro správu univerzitních dokumentů, protože ta současná se pomalu stává nevyhovující a některé velmi staré části zdrojového kódu s sebou přináší řadu omezení, čímž komplikují vývoj. Nasazení nové verze by mělo proběhnout až po plánované aktualizaci celého serveru, díky které se nám naskytla příležitost značně inovovat použité technologie. Největší výhody s sebou přineslo upgrade na Java 8 a Apache Tomcat 9, což umožnilo vyvíjet aplikaci s pomocí aktuálního vydání Spring Boot 2 a modulů ze Spring Framework.

S kolegou jsme se dohodli, že nejprve přepíšu aplikaci se zachováním původní funkcionality a až následně implementuji vylepšení. V rámci modernizace designu stránek InNET bude také vyměnen současný vzhled (obrázek 5) za nový univerzitní vizuál. Po dokončení úkolu v plném rozsahu by pak mělo dojít k zpřehlednění uživatelského rozhraní a výraznému zrychlení při práci s dokumenty, čehož dosáhneme eliminací zbytečného čekání na dokončení operací při komunikaci s úložištěm Alfresco. Z menších plánovaných změn bych jmenoval například rozšířené možnosti vyhledávání a filtrování či opravu nesprávného řazení dle organizační jednotky.

Aplikace v současné podobě poskytuje uživatelům možnost přidávat, editovat a mazat dokumenty včetně příloh. V jednotlivých kategoriích je pak mohou řadit, filtrovat a do jisté míry také prohledávat metadata. Mimo této přímé interakce existují také exporty, které se zapojují jak v InNETu, tak ve veřejném prezentačním webu univerzity. Aplikace také v pravidelných intervalech kontroluje dostupnost dokumentů na úřední desce.



Obrázek 5: Současný vzhled aplikace pro správu univerzitních dokumentů

Úkol se mi bohužel nepovedlo zcela dokončit před odevzdáním této bakalářské práce, nicméně tyto klíčové části tvořící většinu náplně již hotové jsou:

- autentizace,
- autorizace,
- modelové třídy,
- konvertory reprezentací dat,
- aplikační logika ve službách.

6.6.1 Volba technologií

Původně jsme zamýšleli užití klasického Spring Framework bez autokonfigurační nádstavby. Ukázalo se však, že Spring Boot skutečně nabízí mnoho podstatných výhod, které pomáhají předcházet chybám, šetří čas a zlepšují čitelnost kódu. Jeho všeobecná rozšířenost ve většině aplikací využívajících Spring Framework [14] navíc usnadňuje hledání řešení běžných problémů na internetu. Na návrh konzultanta jsem zabezpečení řešil kompletně pomocí Spring Security.

Pro sestavení uživatelského rozhraní se nově přikláníme k zavedení Thymeleaf (tzv. „template engine“), přičemž stávající technologii JSP si necháváme jako zálohu v případě neočekávaných překážek při vývoji. Toto rozhodnutí jsem po konzultaci s kolegou učinil z důvodu podstatně rozsáhlejší dokumentace Thymeleaf a mnohem lepší podpory pro práci se šablonami. Součástí zadání bylo také využití Spring MVC.

6.6.2 Vytvoření šablony Spring Boot aplikace

Rozhodl jsem se nejprve sestavit obecnou šablonu Spring Boot aplikace zabezpečené pomocí univerzitního SSO, neboť celou tuto část lze v budoucnu využít jako startovací bod a zároveň se tím usnadnil počáteční vývoj a testování. Vzniklá testovací aplikace měla za úkol pouze vynutit přihlášení uživatele, mapovat informace poskytnuté v odpovědi CAS serveru na modelovou třídu a vypsat její obsah.

V této části jsem však narazil na hned několik překážek, které mě neočekávaně zdržely. Nejprve se mi vůbec nedařilo nalézt příčinu nesprávného kódování znaků projevujícího se na všech stránkách. S pomocí kolegy se ukázalo, že užitý návod opomněl nastavit tento parametr u jednoho z filtrů, čímž došlo k ovlivnění všech zbylých částí aplikace. Vzhledem k tomu, že daným filtrem byl validátor CAS ticketů, nenapadlo mě jej z chyby podezřívat, protože by měl zpracovávat textové řetězce obsahující pouze základní znaky.

Druhou překážku tvořila nemožnost dynamicky generovat URL služby pro autentizační SSO server. Spring Security bohužel tuto funkcionalitu ve výchozím stavu nenabízí, nicméně po vyčerpání všech ostatních možností jsem naštěstí nalezl návod s postupem, který úpravou chování několika tříd tento problém řeší sestavením adresy na základě obdrženého HTTP požadavku. Informace z takového zdroje mohou samozřejmě být podvržené, což by za normálních okolností

představovalo bezpečnostní riziko, ale v našem případě tomu tak není, protože předem známe neměnný seznam povolených domén a subdomén, jež může URL obsahovat.

Posledním problémem je absence podpory pro CAS gateway autentizaci ve Spring Security. Tuto překážku se mi odstranit nepodařilo, neboť ji nelze nijak obejít ani nahradit jinou funkcionalitou. Nalezení řešení naštěstí není nezbytné, protože filtr s tímto nastavením se v původní verzi aplikace využívá jen zcela minimálně.

Po ukončení práce na odstranění těchto překážek jsem ještě potřeboval uživateli přidělit Spring Security role na základě těch z LDAP. Nešlo tak ale učinit pouze dle názvu (tedy CN z vrácené cesty), neboť existují shody ve zcela odlišných částech stromové struktury, což by mělo za následek nežádoucí přidělení větších oprávnění. Aby bylo možné sestavit z DN v odpovědi CAS serveru unikátní identifikátor role, spojil jsem nezbytné minimum RDN – v našem případě CN a první OU. Popis struktury LDAP nezbytný k realizaci tohoto řešení mi po dotázání poskytl nadřízený.

6.6.3 Analýza

Původní aplikace se skládala z modelů, vrstvy služeb, celé řady „utility“ tříd a poněkud primitivní implementace návrhového vzoru Front Controller, která na základě HTTP GET parametrů spouštěla požadované operace. Zejména tuto část jsme považovali za nevyhovující, protože se v ní těžko orientuje a znemožňuje snadné rozšiřování a proto bude nahrazena běžným Spring MVC.

Samotné služby obsahovaly většinou jen statické metody a proměnné, což sice práci nijak neztěžovalo (při mapování v TLD souboru ji to dokonce ulehčilo), ale nejedná se o korektní objektový návrh. Pro takto navržené třídy by navíc v budoucnu mohlo být obtížné psát unit testy.

Poněkud krkolomný byl také způsob práce s modely a reprezentací dat, kterou aplikaci „vnucoval“ Alfresco konektor, jejž jsem však přepsat nemohl, neboť by tím nepoměrně narostl objem vyžadované práce a času, nemluvě o potížích s kompatibilitou.

6.6.4 Přepsání vrstvy služeb

Rozhodl jsem se nahradit statické prvky instančními, abych dodržel správný objektový návrh a v budoucnu usnadnil testování. Inicializace všech služeb nyní nově probíhá pomocí dependency injection v konstruktoru a následným voláním metody s pevně danou signaturou a speciální Java EE anotací zajišťující její spuštění ihned po vytvoření instance. Logiku v této vrstvě jsem rozdělil do celkem deseti tříd:

- CategoryService – práce s kategoriemi dokumentů
- ConnectorService – operace s Alfresco konektory
- DocumentService – zpracování dokumentů

- FunctionService – nakládání s funkcemi (ve smyslu funkce jako pracovního zařazení)
- FunctionTypeService – práce s typy funkcí
- IntegratorService – získávání dat z integrátoru
- LanguageService – operace s jazyky
- MessageService – logika lokalizace mimo šablony
- OrganisationUnitService – práce s organizačními jednotkami
- PermissionService – získávání oprávnění z úložiště Alfresco

6.6.5 Konvertory

Vzhledem k nevhodnosti modelových tříd z konektoru jsem se po konzultaci s kolegou rozhodl vytvořit vrstvu konvertorů, které by převáděly externí reprezentaci na vhodnější interní a odstínily tak zbytek aplikace. Většina z nich obsahuje pouze malé množství logiky, avšak často se zde využívá rekurze a konverze typů pro datum. Výjimku představuje převodník modelů organizačních jednotek, kde jsem přistoupil ke sloučení informací ze dvou zdrojů dat – integrátoru (tj. IS EDISON) a úložiště dokumentů Alfresco, čímž se zpřehlednila část služeb. Do budoucna počítáme s tím, že by nové modelové třídy z aplikace mohly sloužit jako inspirace při modernizaci konektoru.

6.6.6 Autentizace exportů

Na rozdíl od zbytku aplikace nejsou exporty zabezpečeny standardním SSO, nýbrž jiným druhem ověření pro komunikaci mezi servery. Z tohoto důvodu vznikla nutnost implementovat vlastní rozšíření standardní funkcionality Spring Security, které by přidalo podporu pro tento autentizační postup.

Při práci na této části úkolu jsem bohužel opět několikrát narazil na nedostatek informací a konkrétních příkladů v dokumentaci, většina postupů uváděná ve zdrojích třetích stran se navíc ukázala být naprosto nesprávná, což mě opět nezanedbatelně zdrželo.

Výsledná implementace obsahuje vlastní autentizační filtr zapojený do Spring Security řetězu, přičemž předání tokenu k ověření konkrétní třídě probíhá dle návrhového vzoru Chain of Responsibility. Objekt poskytovatele zabezpečení po přijetí vyhodnotí podporu pro daný token a pokud je to možné, provede autentizaci a vyvolá událost, na kterou filtr reaguje voláním příslušeného handleru pro úspěch či selhání.

7 Zhodnocení

Individuální odborná praxe naprosto předčila všechna má očekávání a pro budoucí studium i uplatnění na trhu práce ji považuji za zcela klíčovou. Výrazně jsem si rozšířil stávající znalosti a naučil se pracovat s celou řadou nových technologií, přičemž minimálně s několika z nich bych se jinak s největší pravděpodobností nesetkal. Dostalo se mi pochopitelně také velkého počtu příležitostí k udržování a rozšiřování zdrojového kódu psaného někým jiným, což v průběhu studia obvykle není možné.

Zlepšil jsem si i mnoho tzv. měkkých dovedností (anglicky „soft skills“), jmenovitě například práci v týmu, komunikaci či plánování. Oceňuji také zisk alespoň kusých informací o vnitřním chodu větší organizace.

7.1 Uplatněné předchozí znalosti

Kvůli složení svého osobního studijního plánu, zejména v posledním ročníku, jsem využil teoretické znalosti získané v průběhu studia pouze v omezené míře. Nejvíce mi pomohl předmět Algoritmy II zabývající se datovými strukturami a jejich vlastnostmi. Ojediněle se mi také podařilo zužitkovat část zkušeností nabytých při vypracování některých projektů a semestrálních prací. Při modernizaci vzhledu administračních rozhraní našly své uplatnění některé postupy uváděné v teoretické části předmětu Uživatelská rozhraní.

S verzovacím systémem Git jsem se již setkal v minulém zaměstnání a při svých osobních projektech, což považuji za nezanedbatelnou výhodu, protože v průběhu studia s ním vůbec nepřijdeme do styku. Díky těmto předchozím zkušenostem mi nebyly zcela cizí ani některé další technologie využívané pro vývoj webových aplikací (například HTML a CSS).

7.2 Chybějící znalosti

Na začátku praxe jsem neměl žádné znalosti rozšíření Java EE ani konkrétní implementace důležitých částí standardní knihovny jazyka Java. Chybělo mi také alespoň letmé seznámení s principem classloaderů (nebo modulů) a některými běžně užívanými technologiemi (například Maven či Gradle). Předmět Správa operačních systémů si bohužel nešlo zapsat dříve než v posledním semestru, což považuji za nešťastné, protože v něm poskytnuté informace mi při řešení úkolů mohly pomoct, kdybych jimi disponoval dříve.

8 Závěr

Tato bakalářská práce si kladla za cíl především popsat celkový průběh mé individuální odborné praxe a jednotlivé řešené úkoly. V úvodní části jsem se věnoval také použitým technologiím, zaměření Centra informačních technologií VŠB-TUO a své motivaci pro výběr tohoto zaměstnavatele. Text je na vhodných místech pro názornost doplněn několika obrázky a jedním výpisem zdrojového kódu. Na konci stručně shrnuji nově nabyté dovednosti a využití těch stávajících.

Z praktického hlediska lze za výsledek mé individuální odborné praxe považovat dokončení implementace popisovaných úkolů, úspěšné otestování změn a jejich nasazení do produkčního prostředí.

Literatura

1. *Statut Centra informačních služeb Vysoké školy báňské – Technické univerzity Ostrava* [online] [cit. 2020-02-09]. Dostupné z: https://dokumenty.vsb.cz/docs/files/cs/37ae208a-ec68-41eb-852f-625f6bbcc23b.
2. *TIOBE Index* [online] [cit. 2020-02-15]. Dostupné z: https://www.tiobe.com/tiobe-index/.
3. *Apache Tomcat* [online] [cit. 2020-02-23]. Dostupné z: http://tomcat.apache.org/.
4. *Maven - Introduction* [online] [cit. 2020-02-23]. Dostupné z: https://maven.apache.org/what-is-maven.html.
5. *Open Hub - Compare Repositories* [online] [cit. 2020-02-23]. Dostupné z: https://www.openhub.net/repositories/compare.
6. *Apache Solr - Features* [online] [cit. 2020-02-15]. Dostupné z: https://lucene.apache.org/solr/features.html.
7. *CAS Protocol 3.0 Specification* [online] [cit. 2020-02-23]. Dostupné z: https://apereo.github.io/cas/4.2.x/protocol/CAS-Protocol-Specification.html.
8. *Alfresco Platform* [online] [cit. 2020-02-16]. Dostupné z: https://www.alfresco.com/platform.
9. *Spring Framework - Features* [online] [cit. 2020-02-16]. Dostupné z: https://spring.io/projects/spring-framework.
10. *Spring - Web on Servlet Stack* [online] [cit. 2020-02-16]. Dostupné z: https://docs.spring.io/spring/docs/5.2.x/spring-framework-reference/web.html.
11. *Spring Security Reference* [online] [cit. 2020-02-16]. Dostupné z: https://docs.spring.io/spring-security/site/docs/5.2.2.RELEASE/reference/htmlsingle/.
12. *Spring Boot - Features* [online] [cit. 2020-02-23]. Dostupné z: https://spring.io/projects/spring-boot.
13. *JVM Ecosystem Report 2020* [online] [cit. 2020-04-16]. Dostupné z: https://snyk.io/wp-content/uploads/jvm_2020.pdf.
14. *The State of Java in 2019* [online] [cit. 2020-04-25]. Dostupné z: https://www.baeldung.com/java-in-2019.